Università degli Studi di Cagliari

Facoltà di Ingegneria

Corso di Laurea Triennale in Ingegneria Elettrica,

Elettronica e Informatica

Creazione di una GUI per il software ps3d

Relatore:

Chiar.mo Prof. Giuseppe Rodriguez

Candidato:

Mattia Urru

matr. 70/89/0372

Anno Accademico 2020/2021

Cagliari - 30/09/2021

Indice

A Mamma e Papà

e ai miei fra.

Capitolo 1

Introduzione

Tra le tecniche fondamentali della Computer Vision,troviamo senz'altro la Photometric Stereo. Essa infatti permette di ricostruire dei modelli 3D a partire da una serie di immagini (quindi, dataset 2D) dell'oggetto che si vuole ricostruire. La tecnica base prevede che di un oggetto vengano prodotte varie immagini, ciascuna con una fonte luminosa (a distanza idealmente infinita) posta ad un angolo noto rispetto al centro del riferimento. Questa tecnica è sintetizzata nel software **ps3d** su cui si basa questa tesi. L'elaborato inizia con una panoramica della tecnica della Photometric Stereo, soffermandosi sulle caratteristiche algoritmiche e sui fondamenti matematici su cui si basa. Nel Capitolo 3 verrà fatto un approfondimento sul software ps3d, concernente le tecnologie utilizzate, gli algoritmi applicati e le metodologie, fino ad arrivare al Capitolo 4 in cui verrà presentato **gps3d**, il software che si propone di fornire un'interfaccia a ps3d permettendone una più facile fruizione. In conclusione, il Capitolo 5 ha l'obbiettivo di tirare le somme del lavoro svolto, con particolare attenzione verso le criticità che hanno condizionato lo sviluppo della soluzione, e con un occhio verso quanto si potrà fare in futuro sia relativamente a ps3d che a gps3d. Questo lavoro si baserà in modo importante sul paper [5]. Un ringraziamento particolare per questo lavoro va al mio relatore, il prof. Giuseppe Rodriguez, che grazie alla sua umanità e competenza è riuscito a guidarmi in questo percorso, rimanendo a disposizione per ogni piccolo dubbio e portando una notevole dose di pazienza.

Capitolo 2

La tecnica della Photometric Stereo

La Photometric Stereo ha trovato il suo spazio negli ambienti della visione computerizzata sicuramente per il suo costo contenuto e la sua semplicità di esecuzione: infatti, rispetto ad altre tecniche, consente di utilizzare una sola fotocamera. Per la prima volta è stata introdotta da R.J. Whoodham nel 1980 [1]. Preliminarmente è necessario fare delle opportune ipotesi sotto cui ipotizziamo l'applicazione di questa tecnica:

- la superficie da ricostruire deve essere Lambertiana [4];
- assumiamo che le fonti luminose siano poste a distanza infinita dall'oggetto;
- non c'è nessun'area della superficie completamente in ombra in tutte le fotografie;
- La fotocamera è posta a una distanza tale per cui le aberrazioni dovute alla prospettiva sono trascurabili.

2.1 Fondamenti teorici

Di seguito verrà data una definizione rigorosa degli strumenti matematici utilizzati, per riuscire a giustificare i risultati ottenuti dal software.

2.1.1 Metodo alle differenze finite

Per approssimare numericamente il valore della derivata di una funzione f in un punto x_0 si ricorre spesso al *metodo alle differenze finite*. Dalla definizione di derivata, dovremmo conoscere un numero infinito di valori che f assume nell'intorno di x_0, ma grazie a questo metodo possiamo utilizzare rapporti incrementali (appunto, differenze finite) in luogo di limiti, ovvero differenze infinitesime.

Poniamo di avere una generica equazione differenziale alle derivate parziali *ellittica*, quindi nella forma

$$u_{xx} + u_{yy} + p(x,y)u_x + q(x,y)u_y + r(x,y)u + s(x,y) = 0$$

che poniamo sia definita in un dominio $\Omega = [a,b] \times [c,d]$ e con $p(x,y), q(x,y), r(x,y), s(x,y)$ continue

Poichè le condizioni sono poste sulla *frontiera* del dominio allora ci troviamo a parlare di un *problema di Dirichlet*. Scritto in notazione matematica:

$$\begin{cases} u_{xx} + u_{yy} + p(x,y)u_x + q(x,y)u_y + r(x,y)u + s(x,y) = 0 \\ u(x,y) = g_1(x,y) \quad (x,y) \in \partial\Omega \end{cases}$$

Possiamo permetterci di fare l'assunzione che nel nostro problema il dominio Ω sia costante, perchè nella Photometric Stereo la fotocamera è sempre nella stessa posizione, quindi la regione di spazio interessata è costante. Applichiamo come segue il metodo alle differenze finite per la risoluzione del problema di Dirichlet: Per prima cosa si discretizza Ω in questo modo:

$$x_i = a + ih \text{ posto } i = 0, 1, ..., n+1 \text{ e } h = \tfrac{(b-a)}{(n-1)}$$
$$y_i = b + jk \text{ posto } j = 0, 1, ..., m+1 \text{ e } k = \tfrac{(d-c)}{(m-1)}$$

Una volta ottenuto un dominio *discreto*, possiamo discretizzare anche le derivate della funzione. Partendo dalla derivata prima, essa si approssima per valori precedenti e successivi (approssimazione mediante differenze centrali) in questo modo:

$$\frac{\partial f(x_i, y_j)}{\partial x} \simeq \frac{f(x_{i+1}, y_j) - f(x_{i-1}, y_j)}{2h}$$

di conseguenza, possiamo approssimare anche la derivata seconda in questa maniera:

$$\frac{\partial^2 f(x_i, y_j)}{\partial x^2} \simeq \frac{1}{h}(\frac{f(x_{i+1}, y_j) - f(x_i, y_j)}{h} - \frac{f(x_i, y_j) - f(x_{i-1}, y_j)}{h})$$

e quindi

$$\frac{\partial^2 f(x_i, y_j)}{\partial x^2} \simeq \frac{f(x_{i+1}, y_j) - 2f(x_i, y_j) + f(x_{i-1}, y_j)}{h^2}$$

questa, è nota come *differenza centrale del secondo ordine* e possiamo affermare che

$$\frac{\partial^2 f(x_i, y_j)}{\partial x^2} - \frac{f(x_{i+1}, y_j) - 2f(x_i, y_j) + f(x_{i-1}, y_j)}{h^2} = O(h^2)$$

Il procedimento appena illustrato era relativo alla variabile x, analogamente si può operare relativamente alla variabile y. Per non appesantire la notazione, ora poniamo $u_{i,j} = u(x_i, y_j)$, $p_{i,j} = p(x_i, y_j)$, $q_{i,j} = q(x_i, y_j)$, $r_{i,j} = r(x_i, y_j)$, , $g_{i,j} = g_1(x_i, y_j)$, quindi possiamo scrivere:

$$\frac{u_{i-1,j} - 2u_{i,j} + u_{i+1,j}}{h^2} + \frac{u_{i,j-1} - 2u_{i,j} + u_{i,j+1}}{k^2} + p_{i,j}\frac{u_{i+1,j} - u_{i-1,j}}{2h} + q_{i,j}\frac{u_{i,j+1} - u_{i,j-1}}{2k} + r_{i,j}u_{i,j} + s_{i,j} = 0$$

con

$$i = 0, 1, 2, ..., n \text{ e } j = 0, 1, 2, ..., m$$

Ordinando:

$$c_{i,j}u_{i,j-1} + b_{i,j}u_{i-1,j} - 2a_{i,j}u_{i,j} + \tilde{b_{i,j}}u_{i+1,j} + \tilde{c_{i,j}}u_{i,j+1} = 2h^2k^2s_{i,j}$$

con

$i = 0, 1, 2, ..., n$ e $j = 0, 1, 2, ...m$ e

- $c_{i,j} = h^2(2 - kq_{i,j})$
- $b_{i,j} = k^2(2 - hp_{i,j})$
- $a_{i,j} = 2(h^2 + k^2) - h^2k^2r_{i,j}$
- $\tilde{b_{i,j}} = k^2(2 + hp_{i,j})$
- $\tilde{c_{i,j}} = h^2(2 + kq_{i,j})$

Detto questo, possiamo affermare che discretizzando la PDE siamo arrivati alla risoluzione di un sistema lineare *m x n* che può essere risolto in modo semplice utilizzando un qualsiasi metodo.

Per approfondimenti si rimanda a [3]

2.1.2 Decomposizione in valori singolari (SVD)

La decomposizione in valori singolari o *single value decomposition* (da qui SVD)è una fattorizzazione di una matrice che generalizza la decomposizione in termini di autovalori e autovettori. Sia data $A \in \mathbb{C}^{m \times n}$ possiamo scrivere

$$A = U\Sigma V^*$$

in cui U è una matrice unitaria $m \times m$ le cui colonne sono dette *vettori singolari sinistri*, Σ una matrice diagonale rettangolare $m \times n$ i cui elementi diagonali sono detti *valori singolari* e V^* la trasposta coniugata di una matrice unitaria V di dimensione $n \times n$ le cui colonne sono dette *vettori singolari destri*.

Possiamo verificare che i vettori singolari sinistri sono gli autovettori di AA^*, i vettori singolari destri sono gli autovettori di A^*A e i valori sulla diagonale di Σ sono le radici quadrate degli autovalori non nulli di AA^* e A^*A

2.1.3 Legge di Lambert

Se, come precedentemente scritto, facciamo l'assunto che la superficie da ricostruire sia *Lambertiana*, possiamo dire che l'intensità luminosa misurata in ciascun punto è proporzionale all'angolo tra la normale alla superficie osservata in quel punto e la direzione della fonte luminosa, e chiamiamo questo assunto *legge del coseno di Lambert*. In notazione matematica possiamo scrivere:

$$\rho(x, y)\, \langle \boldsymbol{n}(x, y), \boldsymbol{l} \rangle = \mathcal{I}(x, y)$$

con $t = 1, ..., q$. La funzione $\rho(x, y)$ rappresenta l'*albedo* in un dato punto della superficie, $\mathcal{I}(x, y)$ è l'intensità luminosa,$\langle \cdot, \cdot \rangle$ è il prodotto scalare in $\mathbb{R}^3$, $\boldsymbol{l}$ è il vettore delle fonti luminose

e $\boldsymbol{n}(x, y)$ è la normale nel punto di coordinate x, y Quando l'albedo è costante, ci troviamo davanti a un cosidetto *riflettore Lambertiano*.

2.2 Algoritmo

Descriveremo ora l'algoritmo *alla Poisson* per la ricostruzione delle superfici. Il primo step consta nell'estrarre le informazioni fotometriche dell'oggetto, ottenendo quindi il campo delle normali, discretizzando la *legge di Lambert* su una griglia regolare, ottenendo il vettore normale alla superficie risolvendo un'equazione matriciale. Il secondo step prevede di utilizzare i dati fotometrici raccolti e, risolvendo un problema differenziale (risolvendo appunto, un'equazione di Poisson), ricostruire la superficie vera e propria. L'equazione di Poisson si ottiene approssimando numericamente la divergenza del campo normale, ottenendo un operatore di Laplace discretizzato. Ci focalizzeremo su un algoritmo che preveda fonti luminose non note a priori.

Per approfondimenti relativi all'algoritmo, si invita ad approfondire su [5]

2.2.1 Assunzioni preliminari

L'oggetto da osservare è posto al centro di un riferimento in $\mathbb{R}^3$, l'asse di osservazione (l'asse z) è diretto dall'oggetto a una telecamera fissa. Il punto di osservazione è a una distanza infinita dall'oggetto, e sono scattate diverse fotografie di risoluzione $(r+2) \times (s+2)$ ciascuna con una fonte luminosa direzionata in modo differente dalla precedente. La lunghezza **reale** dell'asse orizzontale di ciascun'immagine è A, mentre del lato verticale, sotto l'ipotesi che i pixel siano quadrati, è $B = (s+1)h$ con $h = A/(r+1)$. ogni immagine definisce un dominio $\Omega = [-A/2, A/2] \times [-B/2, B/2]$ discretizzato come visto nel paragrafo dedicato al metodo alle differenze finite. La superficie dell'oggetto è rappresentata da una funzione in due variabili $z = u(x, y)$ con $(x, y) \in \Omega$.

2.2.2 I Step: Ottenimento del campo delle normali

Possiamo scrivere la legge di Lambert discretizzata come

$$DN^T L = M$$

in cui

- $D = diag(\rho_1, \rho_2, ..., \rho_p) \in \mathbb{R}^{p \times p}$ è l'albedo,
- $N = [\boldsymbol{n_1}, \boldsymbol{n_2}, ..., \boldsymbol{n_p}] \in \mathbb{R}^{3 \times p}$ è il campo delle normali,
- $L = [\boldsymbol{l_1}, \boldsymbol{l_2}, ..., \boldsymbol{l_q}] \in \mathbb{R}^{3 \times q}$ è la matrice contenente le direzioni delle fonti luminose,
- $M = [\boldsymbol{m_1}, \boldsymbol{m_2}, ..., \boldsymbol{m_q}] \in \mathbb{R}^{p \times q}$ è l'insieme delle immagini, memorizzate per colonne.

Affinchè la soluzione della precedente relazione sia univoca, serve che $q \geq 3$, ovvero che all'algoritmo vengano fornite almeno 3 immagini.

Il problema consiste nel calcolare la fattorizzazione

$$\tilde{N}^T L = M$$

in cui, come vediamo dalla legge di Lambert discretizzata, $\tilde{N} = ND$. Il problema di per sè non ha una soluzione univoca, ma possiamo fare degli assunti fisici per ottenerne una.

La relazione $\tilde{N}^T L = M$ è soddisfatta per ogni coppia $Q\tilde{N}$, QL con $Q \in \mathbb{R}^{3 \times 3}$ ortogonale. Ogni coppia $A^{-T}\tilde{N}$, AL con $A \in \mathbb{R}^{3 \times 3}$ e non singolare soddisfa la precedente relazione. Dal momento che i vettori $\boldsymbol{n_i}$ sono normalizzati, la norma della *i-esima* colonna di $\tilde{N}$ è uguale all'albedo ρ_i, mentre $\|\boldsymbol{l}_t\|$ è proporzionale all'intensità luminosa. Questo implica che A debba essere ortogonale.

Ciò detto, l'orientazione dell'oggetto non può essere univocamente determinata senza ulteriori informazioni, questa ambiguità è detta *ambiguità di bassorilievo* [6].

Non è restrittivo assumere che $\|\boldsymbol{l}_t\| = 1$, con $t = 1, 2, ..., q$

Facendo adesso la SVD della matrice M, abbiamo

$$M = U\Sigma V^T$$

- $\Sigma = diag(\sigma_1, \sigma_2, ..., \sigma_q)$ è la matrice diagonale contenente i valori singolari
- $U \in \mathbb{R}^{p \times p}$ matrice la cui colonna ortonormale $\boldsymbol{u_i}$ è il vettore singolare sinistro

- $V \in \mathbb{R}^{p \times q}$ matrice la cui colonna ortonormale $\boldsymbol{v_i}$ è il vettore singolare destro

Nel nostro caso di studi, q è molto minore di p, perchè il numero di pixel nelle immagini è molto grande. Come già visto, l'unica condizione è che $q \geq 3$ affinchè si possa trovare una soluzione al problema.

Nel caso reale, le immagini non sono prese in condizioni ideali e potrebbe esservi presente del rumore fotografico, quindi la SVD di M potrebbe avere un rango $r > 3$. Dobbiamo perciò eseguire una SVD troncata, tale che $\sigma_4 = \cdots = \sigma_q = 0$. Poniamo $W = [\sigma_1 \boldsymbol{u_1}, \sigma_2 \boldsymbol{u_2}, \sigma_3 \boldsymbol{u_3}]^T$ e $Z = [\boldsymbol{v_1}, \boldsymbol{v_2}, \boldsymbol{v_3}]^T$ così che $W^T Z \approx M$

Vediamo ora come (e perchè) è possibile determinare il campo normale e le posizioni delle fonti luminose tramite la legge di Lambert discretizzata utilizzando almeno 6 immagini.

Consideriamo la fattorizzazione $W^T Z = M$, posta la semplificazione $\|\boldsymbol{l}_t\| = 1$, determiniamo una matrice B tale che $\|B\boldsymbol{z}_t\| = 1$ per ogni t da 1 a q. Ciò vuol dire risolvere un sistema lineare del tipo $diag(Z^T G Z) = \boldsymbol{e}$ con $\boldsymbol{e} = (1, ..., 1)^T \in \mathbb{R}^q$ e $G = B^T B \in \mathbb{R}^{3 \times 3}$ simmetrica e definita positiva. Poniamo che G dipenda da 6 parametri $g_{i,j}$ con $i \leq j$. Ogni equazione del sistema precedentemente descritto sarà del tipo

$$\boldsymbol{z_t}^T G \boldsymbol{z_t} = \sum_{i,j=1}^{3} z_{it} z_{jt} g_{ij} = 1$$

Il sistema può essere riscritto, in maniera più elegante come

$$H\boldsymbol{g} = \boldsymbol{e}$$

con $\boldsymbol{g} = (g_{11}, g_{22}, g_{33}, g_{12}, g_{13}, g_{23})^T$, $\boldsymbol{e} = (1, \cdots, 1)^T$e $H \in \mathbb{R}^{q \times 6}$ le cui colonne sono

$$[z_{1t}^2, z_{2t}^2, z_{3t}^2, 2z_{1t}z_{2t}, 2z_{1t}z_{3t}, 2z_{2t}z_{3t}]$$

con $t = 1, ..., q$

Si nota come la condizione necessaria affinchè $\boldsymbol{g}$ sia univoco è che $q \geq 6$

Notiamo però che è una condizione necessaria, ma non *sufficiente*. Infatti H potrebbe ancora non soddisfare la condizione sul rango anche con $q \geq 6$.

Rappresentiamo ora B con la sua *fattorizzazione QR*, $B = QR$. R può essere ottenuto dalla *decomposizione di Cholesky* [7] $G = R^T R$ mentre non esiste un modo univoco di determinare Q (Come abbiamo visto in precedenza)

La matrice Q può essere scelta arbitrariamente secondo i criteri descritti nella sezione V di [5]

Una volta determinata Q, il problema si può risolvere come segue:

$$\tilde{N} = QR^{-T}W$$

$$L = QRZ$$

abbiamo così ottenuto il campo delle normali e il vettore delle fonti luminose.

2.2.3 II Step: risoluzione del problema differenziale

Una volta ottenuto il campo normale, consideriamo i vettori

$$[(u_x)_k, (u_y)_k, -1]^T = \frac{\boldsymbol{n_k}}{(n_k)_3}$$

ottenuto normalizzando a -1 la componente con indice 3 del vettore $\boldsymbol{n_k}$. Deriviamo numericamente le componenti di indice 1 e 2 rispettivamente rispetto a x e y per ottenere un'approssimazione sulla griglia di pixel del Laplaciano $\nabla^2(x, y) = u_{xx} + u_{yy}$. Per approssimarlo, utilizziamo un'approssimazione alle differenze finite centrata al secondo ordine, ottenendo

$$f_{ij} = f(x_i, y_j) \approx \frac{(u_x)_{i+1,j} - (u_x)_{i-1,j}}{2h} + \frac{(u_y)_{i,j+} - (u_j)_{i,j-1}}{2h}$$

La superficie (che, come detto in precedenza, è rappresentata da $z = u(x, y)$) può essere ottenuta risolvendo la PDE di Poisson che segue:

$$\nabla^2 u(x, y) = f(x, y)$$

a cui bisogna porre delle opportune condizioni di vincolo al fine di ottenere una soluzione univoca. Discretizziamo l'equazione di Poisson usando un metodo alle differenze finite del second'ordine, e poniamo delle opportune condizioni di Dirichlet, che considerando il dominio $\Omega \in [-A/2, A/2] \times [-B/2, B/2]$ possono essere espresse come segue:

$$u(x, -B/2) = \phi_1(x),\ u(x, B/2) = \phi_2(x) \quad x \in [-A/2, A/2],$$

$$u(-A/2, Y) = \psi_1(y),\ u(A/2, y) = \psi_2(y) \quad y \in [-B/2, B/2].$$

Ora, per semplificare la notazione poniamo $u_{ij} = u(x_i, y_j)$ e $f_{ij} = f(x_i, y_j)$ in ciascun punto della griglia di pixel. Le condizioni al contorno sono quindi $u_{i,0} = \phi_1(x_i)$, $u_{i,s+1} = \phi_2(x_i)$, $u_{0,j} = \psi_1(y_j)$, $u_{r+1,j} = \psi_2(y_j)$.

Discretizziamo la PDE di Poisson con uno schema alle differenze finite a 5 punti con passo h e otteniamo

$$u_{i-1,j} + u_{i,j-1} - 4u_{i,j} + u_{i,j+1} + u_{i+1,j} = \tilde{f}_{i,j}$$

con $i = 1, ..., r$, $j = 1, ..., s$ e $\tilde{f}_{i,j} = h^2 f_{i,j}$ Questa è un'approssimazione di ordine $O(h^2)$.

Aggregando i punti u_{ij} per colonne, otteniamo un sistema del tipo

$$\begin{cases} T\boldsymbol{u_1} + I_s\boldsymbol{u_2} = \boldsymbol{b_1} \\ I_s\boldsymbol{u}_{i-1} + T\boldsymbol{u}_i + I_s\boldsymbol{u}_{i+1} = \boldsymbol{b_i} \\ I_s\boldsymbol{u}_{r-1} + T\boldsymbol{u}_r = \boldsymbol{b}_r \end{cases} . \tag{2.1}$$

dove $i = 2, ..., r-1$ e I_s è la matrice identità $s \times s$. Per quanto riguarda il primo membro, abbiamo:

$$T = \begin{bmatrix} -4 & 1 & & \\ 1 & -4 & \ddots & \\ & \ddots & \ddots & 1 \\ & & 1 & -4 \end{bmatrix}$$

con $T \in \mathbb{R}^{s \times s}$, e

$$\boldsymbol{u}_i = \begin{bmatrix} u_{i,1} \\ u_{i,2} \\ \vdots \\ u_{i,s} \end{bmatrix}$$

con $\boldsymbol{u}_i \in \mathbb{R}^s$ e $i = 1, ..., r$.

Considerando invece il secondo membro:

$$\boldsymbol{b}_1 = \begin{bmatrix} \tilde{f}_{1,1} \\ \tilde{f}_{1,2} \\ \vdots \\ \tilde{f}_{1,s} \end{bmatrix} - \begin{bmatrix} u_{0,1} + u_{1,0} \\ u_{0,2} \\ \vdots \\ u_{0,s} + u_{1,s+1} \end{bmatrix},$$

$$\boldsymbol{b}_r = \begin{bmatrix} \tilde{f}_{r,1} \\ \tilde{f}_{r,2} \\ \vdots \\ \tilde{f}_{r,s} \end{bmatrix} - \begin{bmatrix} u_{r+1,1} + u_{r,0} \\ u_{r+1,2} \\ \vdots \\ u_{r+1,s} + u_{r,s+1} \end{bmatrix},$$

$$\boldsymbol{b}_i = \begin{bmatrix} \tilde{f}_{i,1} \\ \tilde{f}_{i,2} \\ \vdots \\ \tilde{f}_{i,s} \end{bmatrix} - \begin{bmatrix} u_{i,0} \\ 0 \\ \vdots \\ 0 u_{i,s+1} \end{bmatrix}, \ i = 2, ..., r-1,$$

con $\boldsymbol{b}_1, \boldsymbol{b}_r, \boldsymbol{b}_i \in \mathbb{R}^s$

Il sistema può essere rappresentato nella forma compatta

$$A\boldsymbol{u} = \boldsymbol{b},$$

con

$$\boldsymbol{u} = \begin{bmatrix} \boldsymbol{u}_1 \\ \boldsymbol{u}_2 \\ \vdots \\ \boldsymbol{u}_r \end{bmatrix} \in \mathbb{R}^p,$$

$$\boldsymbol{b} = \begin{bmatrix} \boldsymbol{b}_1 \\ \boldsymbol{b}_2 \\ \vdots \\ \boldsymbol{b}_r \end{bmatrix} \in \mathbb{R}^p,$$

e

$$A = \begin{bmatrix} T & I_s & & \\ I_s & T & \ddots & \\ & \ddots & \ddots & I_s \\ & & I_s & T \end{bmatrix} \in \mathbb{R}^{p \times p},$$

Che è un sistema lineare facilmente risolvibile.

Capitolo 3

Panoramica sul software ps3d

Il software, descritto nel paper [5], è una collezione di routines in MATLAB. Il pacchetto è scaricabile da `http://bugs.unica.it/gppe/software/ps3d/` alla data di stesura di questo elaborato l'ultima versione è la 1.0 di Giugno 2021. Il pacchetto è stato scritto utilizzando MATLAB 2021 Il software contiene sia le routines per trattare dati reali, sia quelle utilizzabili per creare dei dati sintetici (quindi con caratteristiche note) al fine di valutare la bontà delle informazioni fornite dal software.

Nello specifico, il pacchetto originale si compone dei seguenti script .m, di seguito illustrati con il loro nome e la loro funzione (verrà altresì dedicato un paragrafo per ciascuno script critico nel flusso di ricostruzione con dati reali):

- `ps3d.m` - Routine principale, *core* del pacchetto. Attraverso essa sono richiamati `ps3dstep1.m` e `ps3dstep2.m` e vengono prodotti tutti i dati fondamentali necessari alla ricostruzione della superficie 3D.
- `ps3dstep1.m` - Svolge il primo step di ricostruzione indicato nel paragrafo 2.2.2, ovvero l'ottenimento del campo normale, dell'albedo e del vettore delle fonti luminose.
- `ps3dstep2.m` - Svolge il secondo step di ricostruzione indicato nel paragrafo 2.2.3, ovvero la discretizzazione del problema differenziale di Poisson e la risoluzione del sistema lineare associato, restituendo le coordinate della superficie da ricostruire.

Questi erano le tre funzioni di interfaccia, che raggruppano le varie operazioni necessarie alla ricostruzione. Per fare ciò, essi si avvalgono di un insieme di funzioni ausiliarie, di seguito indicati:

- `normlights.m` - Normalizza le colonne della matrice dei punti luminosi.
- `poisdir.m` - Risolve un sistema di Poisson definito con delle condizioni al contorno di Dirichlet omogenee.
- `poisneu.m` - Risolve un sistema di Poisson definito con delle condizioni al contorno di Neumann .
- `rotatelights.m` - Restituisce la rotazione delle fonti luminose ricostruite mediante `ps3dstep1.m`.
- `plotlights2d.m` - Crea un grafico 2D delle fonti luminose ricostruite.
- `plotlights3d.m` - Crea un grafico 3D delle fonti luminose ricostruite.

Di seguito invece, una panoramica dei metodi per generare dataset *sintetici*:

- `choosefun.m` - Permette di scegliere tra 5 diverse funzioni in due variabili che descrivono una superficie.
- `createlights.m` - Crea il vettore delle fonti luminose
- `makealbedo.m` - Crea l'albedo
- `makeimage.m` - Crea la matrice delle immagini

Sono presenti anche degli script che realizzano, dall'inizio alla fine, tutto il processo di applicazione della Photometric Stereo su due esempi particolari. Essi sono `testshell.m` e `testsynth.m`

In particolare, utilizzano dei dataset inclusi nel pacchetto `ps3d`: `shell.mat` e `synth.mat`, che contengono le variabili `L` (matrice delle fonti luminose), `M` matrice delle immagini, `r` e `s`

dimensioni dell'immagine, rispettivamente di una conchiglia e di una funzione sintetica generata. Di seguito sono presentati graficamente i dataset utilizzati dai suddetti esempi. Completano il set di script `showshell.m` e `showsinth.m`, usati per visualizzare i dataset di prova.

3.1 Dettaglio degli script fondamentali

Alcuni script concorrono particolarmente al processo di Photometric Stereo, ed è quindi doveroso approfondire il loro comportamento e la loro struttura. Per comprendere a pieno i passaggi effettuati da questi metodi si raccomanda di leggere prima di tutto 2.2.

3.1.1 ps3dstep1.m

La funzione ha i sequenti parametri in ingresso:

- `M` - Matrice delle immagini
- `r` - Larghezza in pixel
- `s` - Lunghezza in pixel
- `light2` - Fonte luminosa alla sinistra (usata come riferimento) - *Opzionale*
- `show` - Flag che determina se vengono o non vengono mostrati i dati ottenuti dal processo (Valori singolari della matrice e posizione delle fonti luminose) - *Opzionale*

La prima operazione saliente è la decomposizione in valori singolari della matrice delle immagini

```
[U1,S1,V1] = svd(M,0);
```

I valori singolari della matrice M sono poi mostrati su un opportuno grafico con un asse semilogaritmico. Successivamente, si ottengono N1 e L1 (Campo normale e Matrice delle fonti luminose pre-normalizzazione

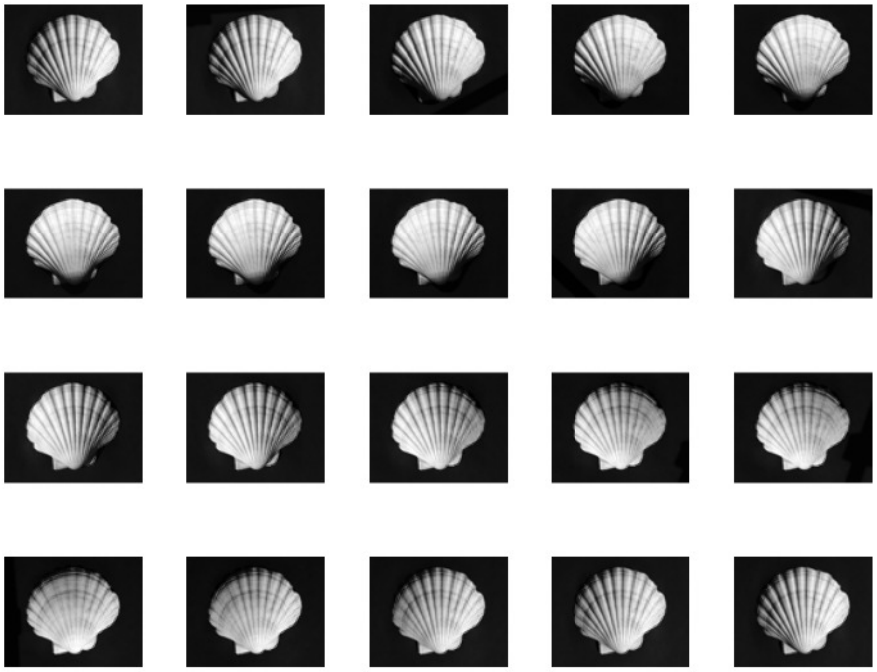

Figura 3.1: Il set di immagini della conchiglia

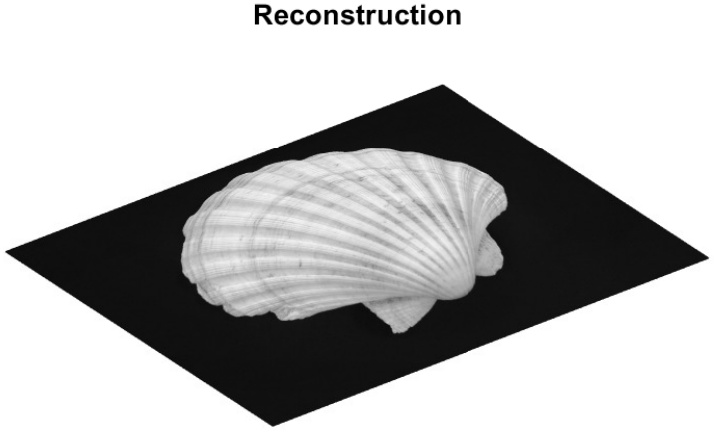

Figura 3.2: La superficie della conchiglia ricostruita da `testshell.m`

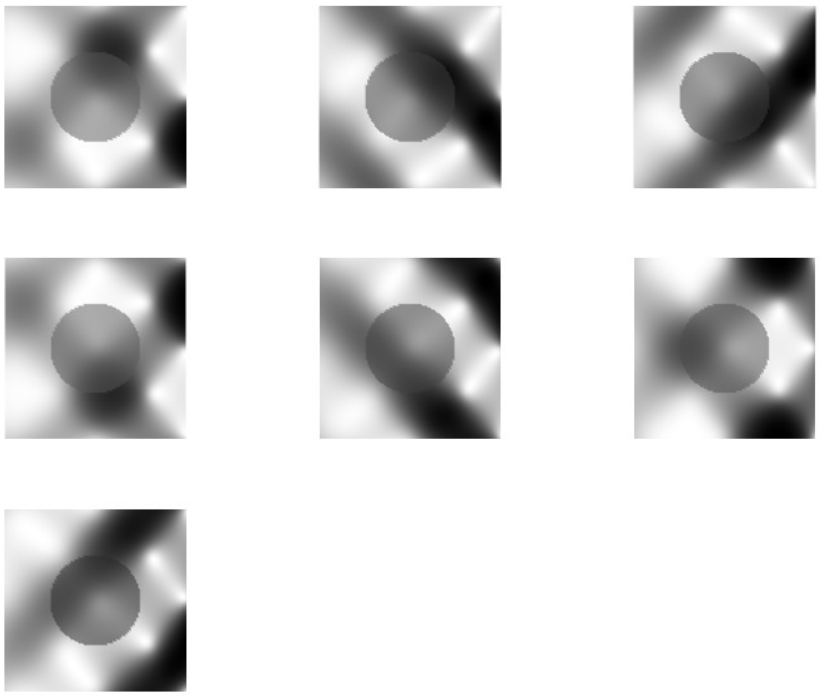

Figura 3.3: Il set di immagini della superficie sintetica

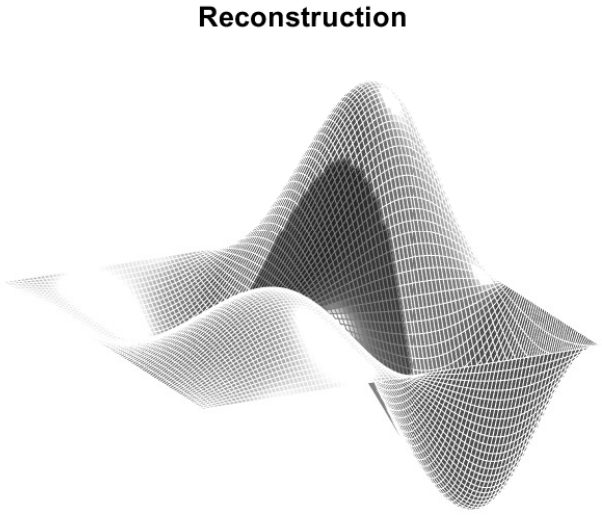

Figura 3.4: La superficie sintetica ricostruita da `testsynth.m`

```
sigma = diag(S1(1:3,1:3));
N1 = U1(:,1:3)';
L1 = diag(sigma) * V1(:,1:3)';
```

Che vengono poi normalizzate con

```
[N2,L2,R,H] = normlights( N1, L1);
```

Le fonti di luce così trovate devono subire un cambio di riferimento, che viene fatto con

```
[NT,L,Qrot] = rotatelights( N2, L2, light2);
```

questa operazione ci restituisce, tra i suoi dati di output, la matrice L definitiva. L'ultimo passaggio che manca è quello che normalizza il campo delle normali e estrae l'albedo (D), e si ottiene con

```
norms = sqrt(sum(NT.2)');
N = NT*spdiags(1./norms,0,p,p);
D = reshape(norms,s,r);
D = D/max(max(D));
```

3.1.2 ps3dstep2.m

La funzione ha i sequenti parametri in ingresso:

- NT - Campo normale normalizzato
- `r` - Larghezza in pixel
- `s` - Lunghezza in pixel
- `bcond` - Parametro che determina, in base al valore (1 o 2) se il problema è modellato con delle condizioni al contorno di Dirichlet o di Neumann - *Opzionale*
- `bdata` - Contiene i dati delle condizioni al contorno di Neumann - *Opzionale*

Una volta ottenuti D, N, L e NT abbiamo tutti i dati per modellare il problema di Poisson e risolverlo.

Come primo passo, si crea il dominio regolare Ω

```
h = A/(r-1);
B = (s-1)*h;
x = linspace(-A/2,A/2,r);
y = linspace(-B/2,B/2,s);
[X,Y] = meshgrid(x,y);
```

In cui `h` è un larghezzza (in pixel) mentre `B` è un'altezza in unità metriche. La funzione `linspace` crea *n* (determinati dall'ultimo parametro) punti equispaziati che vanno dal punto indicato al primo parametro al punto indicato dal secondo parametro. `meshgrid` crea una griglia usando come assi i due vettori `x` e `y` precedentemente creati. Il nostro dominio è quindi `[X,Y]`.

Successivamente, si estraggono i gradienti, contenuti nelle matrici P e Q, dividendo rispettivamente la prima e la seconda riga per la terza riga di NT, e mettendo il risultato in una matrice `r` x `s`

```
P = reshape(-NT(1,:)./NT(3,:),s,r);
Q = reshape(-NT(2,:)./NT(3,:),s,r);
```

In fine, si risolve il problema differenziale, ottenendo la matrice U che definisce la superficie

```
F = zeros(s,r);
F(2:s-1,2:r-1) = (P(2:s-1,3:r)-P(2:s-1,1:r-2)+Q(3:s,2:r-1)-
  ↪ Q(1:s-2,2:r-1))/2/h;
if bcond == 1  % Dirichlet homogeneous boundary conditions
      U = poisdir(F,h);
elseif bcond == 2 % Neumann homogeneous boundary conditions
      if isempty(bdata)
            U = poisneu(F,h);
      else
```

```
            U = poisneu(F,h,bdata-1,bdata-2,bdata-3,bdata-4,
                ↪ bdata-5);
        end
else
        error('Unknown boundary conditions.')
end
```

3.1.3 poisdir.m

La funzione ha i sequenti parametri in ingresso:

- F - Matrice del problema discretizzato
- h - Passo di discretizzazione

Per prima cosa, ottengo le dimensioni del problema

```
[n,m] = size(F);
m = m-2;
n = n-2;
```

Genero poi le matrici `T`, `Im`, `In`

```
%T = sparse(toeplitz([-4 1 zeros(1,n-2)]));
T = spdiags(ones(n,1)*[1 -4 1],[-1,0,1],n,n);
In = speye(n);
Im = speye(m);
%S = sparse(toeplitz([0 1 zeros(1,m-2)]));
S = spdiags(ones(m,2),[-1,1],m,m);
```

Infine, creo la *matrice a blocchi* A, con la quale risolverò il problema vero e proprio, mediante il *prodotto di Kronecker* implementato dalla funzione `kron`

```
A = kron(Im,T)+kron(S,In);
```

L'ultimo passo consiste nella risoluzione vera e propria del problema

```
f = reshape(F(2:n+1,2:m+1),m*n,1);
u = A‘f;
u = h2*u;
U = zeros(n+2,m+2);
U(2:n+1,2:m+1) = reshape(u,n,m);
```

Capitolo 4

Realizzazione di un'interfaccia grafica: gps3d

Il pacchetto **ps3d** è sicuramente una suite molto ampia e permette una vasta gamma di operazioni legate alla Photometric Stereo. Manca però un'interfaccia utente semplice ed efficace: nella versione base di ps3d infatti, l'utente deve invocare manualmente i vari script a mano da riga di comando, oltre che caricare manualmente nel workspace `M`, `r` e `s`. La soluzione a questo problema è data dal software **gps3d**, presentato di seguito.

4.1 L'ambiente di sviluppo AppDesigner

AppDesigner è il tool offerto da MATLAB per la costruzione di interfacce grafiche. Ha sostituito il vecchio GUIDE, ora in via di deprecazione. L'ambiente si presenta come un vero e proprio IDE, e permette la creazione di applicazioni sia con un'interfaccia *drag and drop* sia interagendo con l'ambiente applicativo mediante codice. Il software è integrato in MATLAB. AppDesigner è fortemente *event-oriented*, e basa tutto il flusso sulla definizione di callback, invocate mediante l'interazione dell'utente con gli elementi grafici dell'interfaccia. La libreria standard di AppDesigner comprende una serie di componenti grafici di input come pulsanti, checkboxes, radio buttons, slider, pannelli etc. ma anche di output come immagini, grafici etc.

Le applicazioni create con questo ambiente, sono rappresentate da classi: MATLAB infatti, pur non essendo un linguaggio fortemente tipizzato, supporta il paradigma della *OOP*. Le classi

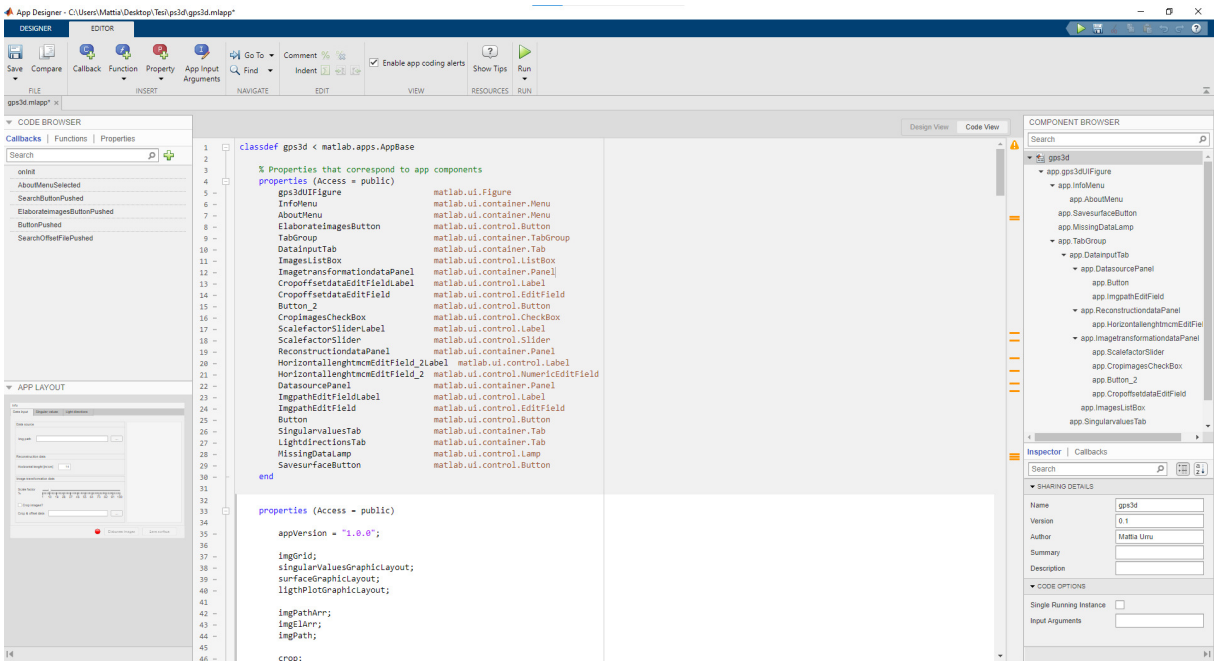

Figura 4.1: L'ambiente di sviluppo AppDesigner

che rappresentano un'applicazione sono figlie della superclasse `matlab.apps.AppBase`. Gli elementi grafici che compongono la UI dell'applicazione sono rappresentati come attributi *pubblici* della classe, così come per le callback, che sono gestite come metodi pubblici.

La creazione di componenti dall'ambiente di interazione visuale comporta l'inserimento automatico del codice necessario alla loro inizializzazione nel metodo privato `createComponents`.

Inoltre è presente una basilare gestione dei *lifecycle hooks* dell'istanza dell'applicativo, nello specifico è possibile definire azioni da fare allo startup dell'applicazione definendo un metodo `onInit`. Nell'immagine 4.1 possiamo vedere come si presenta l'ambiente. La colonna destra è divisa in due parti principali: in alto è presente l'albero gerarchico di tutti gli elementi grafici della vista; selezionandone uno specifico nella parte basse ne vengono mostrati i dettagli. La colonna di sinistra, divisa a sua volta in due sezioni, mostra la parte *Code Browser*, che permette di creare callback, funzioni e attributi della classe rappresentante l'applicazione, mentre nella parte bassa è presente un'anteprima del layout dell'app. La parte centrale dell'ambiente è dedicata al codice vero e proprio.

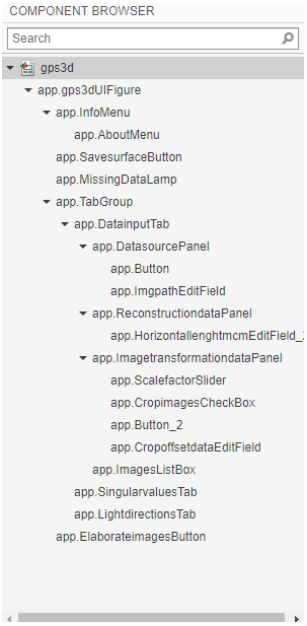

Figura 4.2: Dettaglio dell'albero gerarchico degli elementi grafici

4.2 Metodologie adottate e descrizione della UX e della UI

Il primo passo per definire la *user experience* è stato individuare il flusso di lavoro più logico per un'utente che desiderasse fare una ricostruzione 3D: per prima cosa, deve cercare le immagini attraverso le quali fare l'elaborazione, successivamente deve valutare come impostare la ricostruzione. Opzionalmente, il dataset può essere alterato (Ad esempio, ritagliando le immagini). Si è scelto di realizzare sezionamento e l'inquadramento della UI mediante un layout a Tab. Tutto quello che serve ad iniziare una nuova elaborazione, si trova all'interno del primo tab. quello aperto allo startup. Ovviamente un utente sarà portato a cliccare gli altri tab, che sono opportunamente inizializzati con un messaggio che invita l'utente a iniziare l'elaborazione. Il controllo dei vincoli di integrità sui dati passa attraverso diversi presidi: il primo strato blocca sul nascere comportamenti inattesi, ovvero è impedito all'utente di iniziare una nuova elaborazione disabilitando il relativo bottone finchè non è stata caricata una lista di immagini; il secondo invece effettua un controllo sui dati effettivamente inseriti e in caso essi siano difformi da quanto ci si aspetta viene segnalato un errore all'utente mediante finestre modali. All'utente è inoltre impedito iniziare una nuova elaborazione mentre ne è in corso già una,

al fine di evitare sovraccarichi di lavoro per il computer. Nel corso dell'elaborato verranno mostrati diversi screenshot dell'interfaccia, presentando un'elaborazione d'esempio.

4.3 Flusso di lavoro

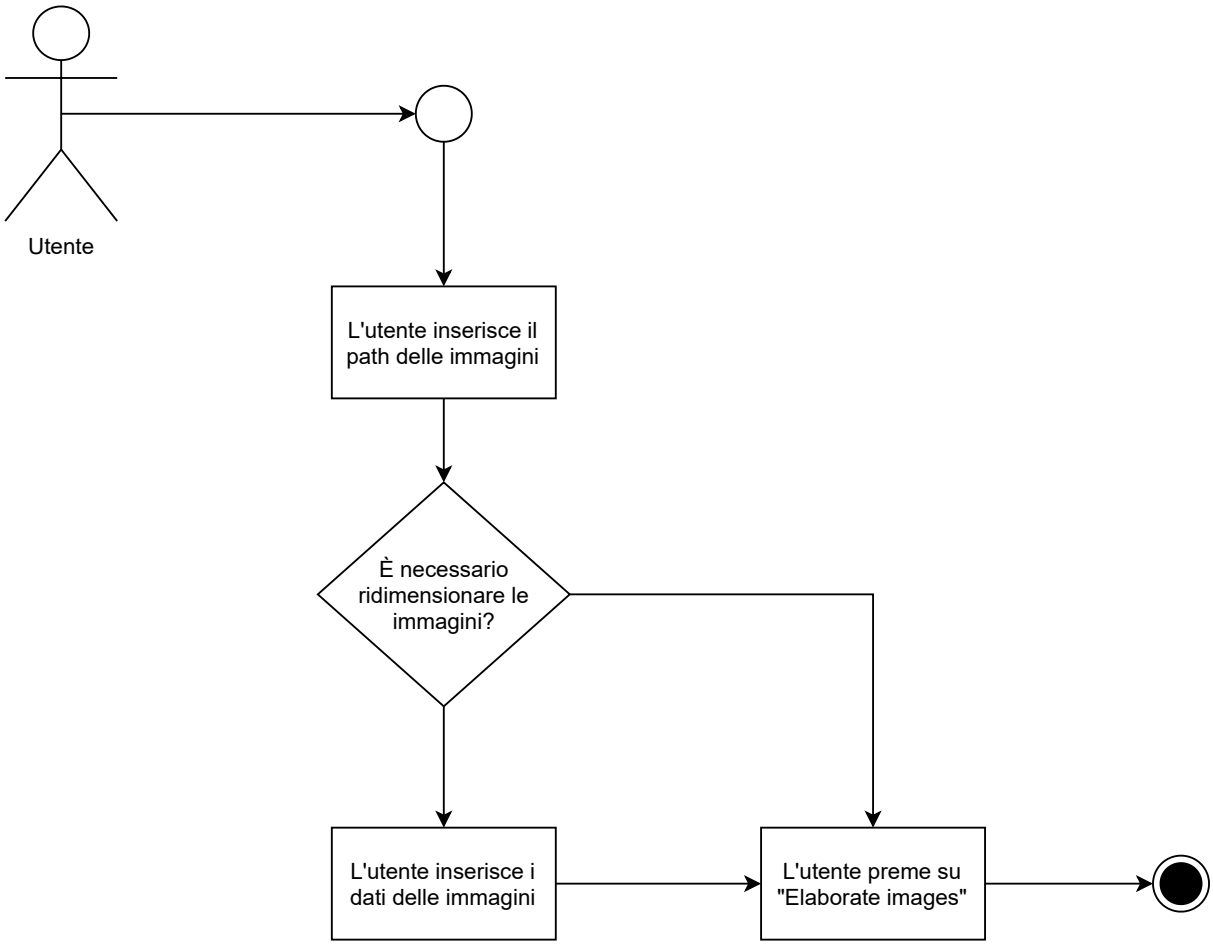

Figura 4.3: Il flusso di lavoro base

L'utente, appena entrato nell'applicazione, ricerca un nuovo dataset di immagini (che per ragioni spiegate in 2.2.2 deve essere composto almeno da 6 elementi. Successivamente, inserisce il valore della lunghezza orizzontale reale (Se diverso dal default) che influenza la ricostruzione. Qualora l'utente desiderasse ridimensionare l'immagine, può farlo caricando un opportuno file mat, che deve contenere un vettore `offset` con numero di elementi pari al

numero di immagini e una matrice `cropMatrix` di dimensione 4×4 che contiene gli angoli dell'area di ridimensionamento. A questo punto, una volta cliccato su "Elaborate images" inizia il vero e proprio processo di ricostruzione: per prima cosa vengono lette da disco le immagini. Successivamente, il dataset letto viene mostrato (in una nuova finestra, per poter agevolmente fare eventuali confronti) come si vede nella 4.6. Per essere utilizzata in modi pratici, la superficie deve essere esportata: premendo su *Save Surface* viene aperto un menu contestuale per il salvataggio del file, che viene esportato in formato *ply*. La superficie può essere visualizzata da un visualizzatore esterno, come possiamo vedere in 4.7

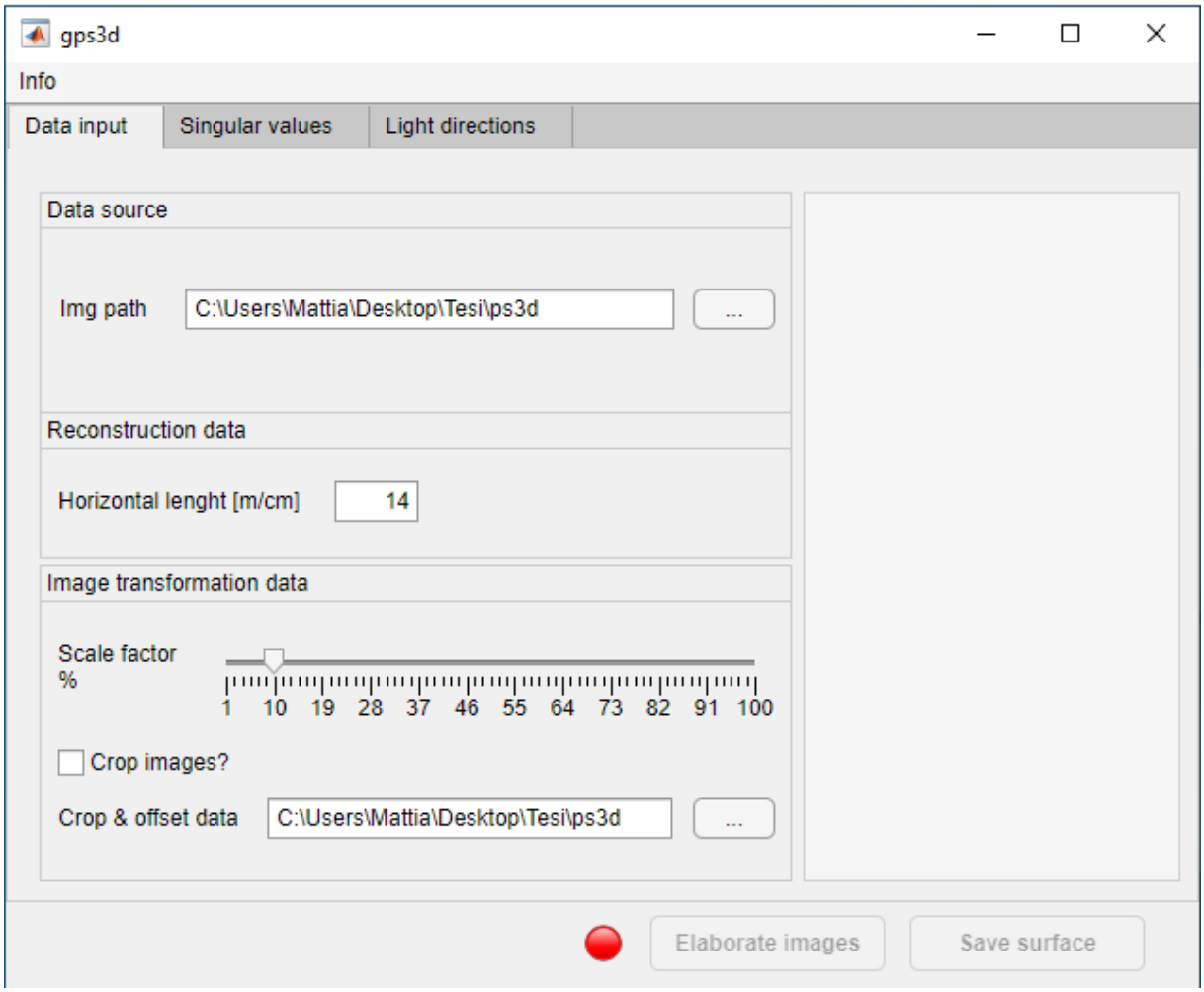

Figura 4.4: L'applicazione, come si presenta allo startup

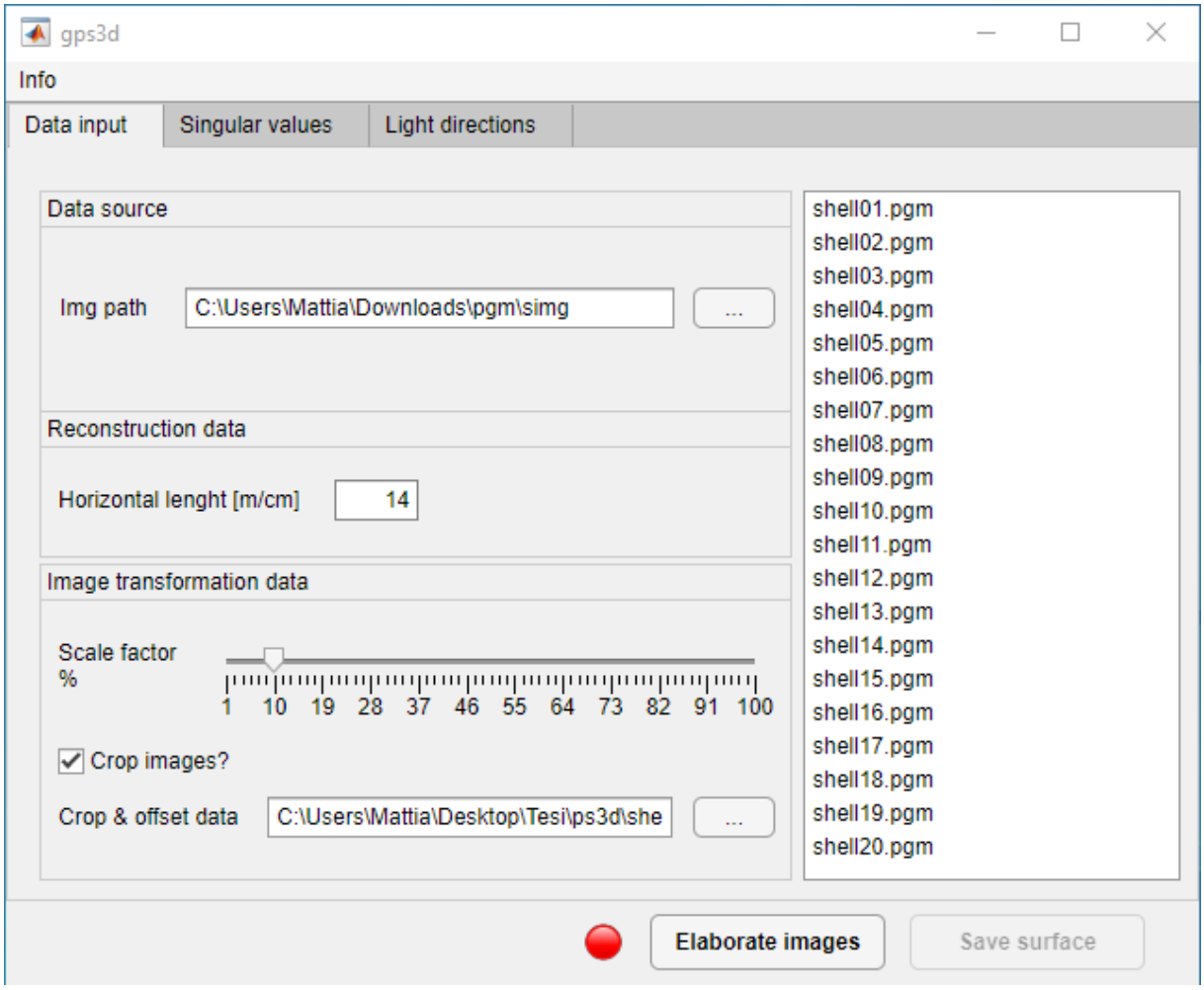

Figura 4.5: Una volta selezionate le immagini, l'app si presenta così

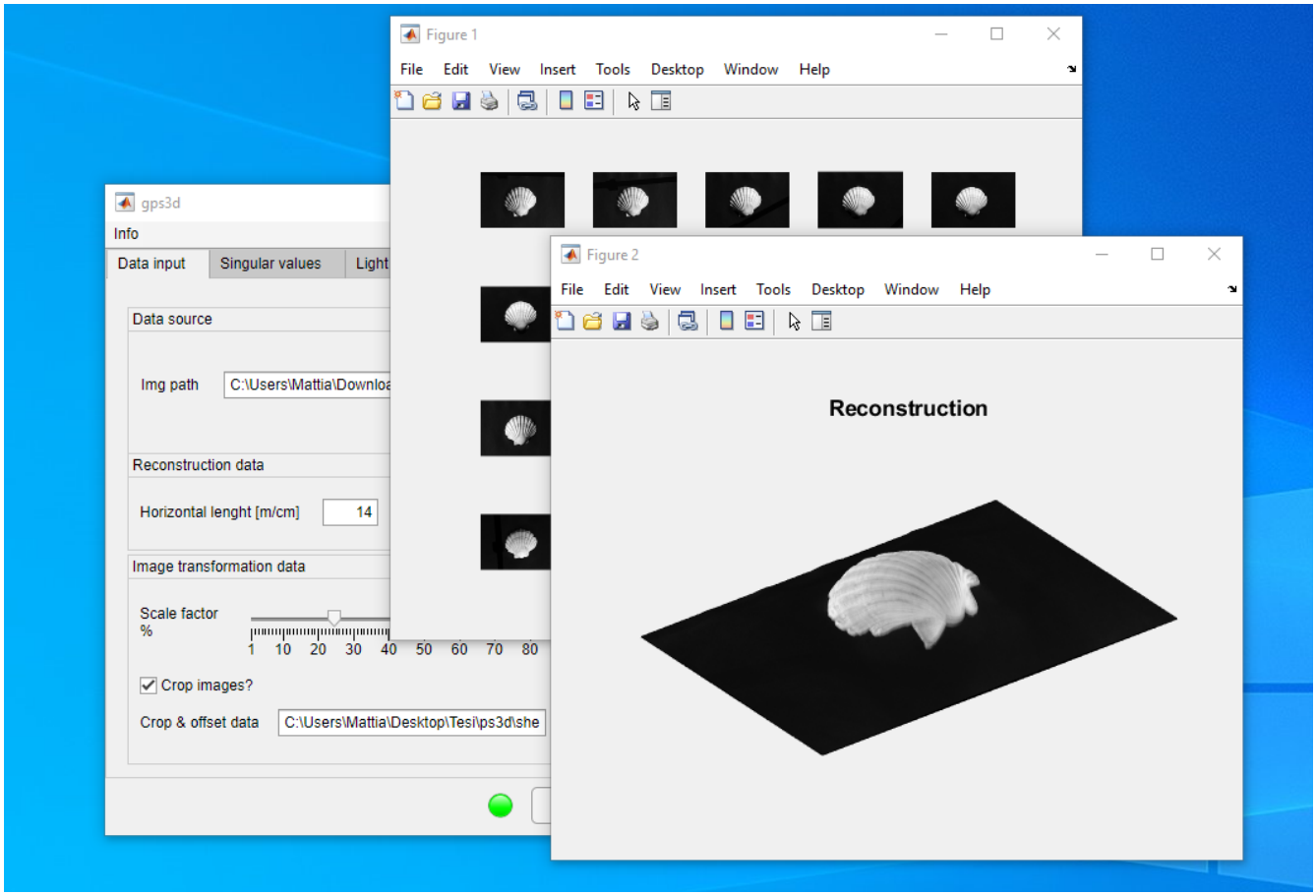

Figura 4.6: Al termine dell'elaborazione, questo sarà lo spazio di lavoro

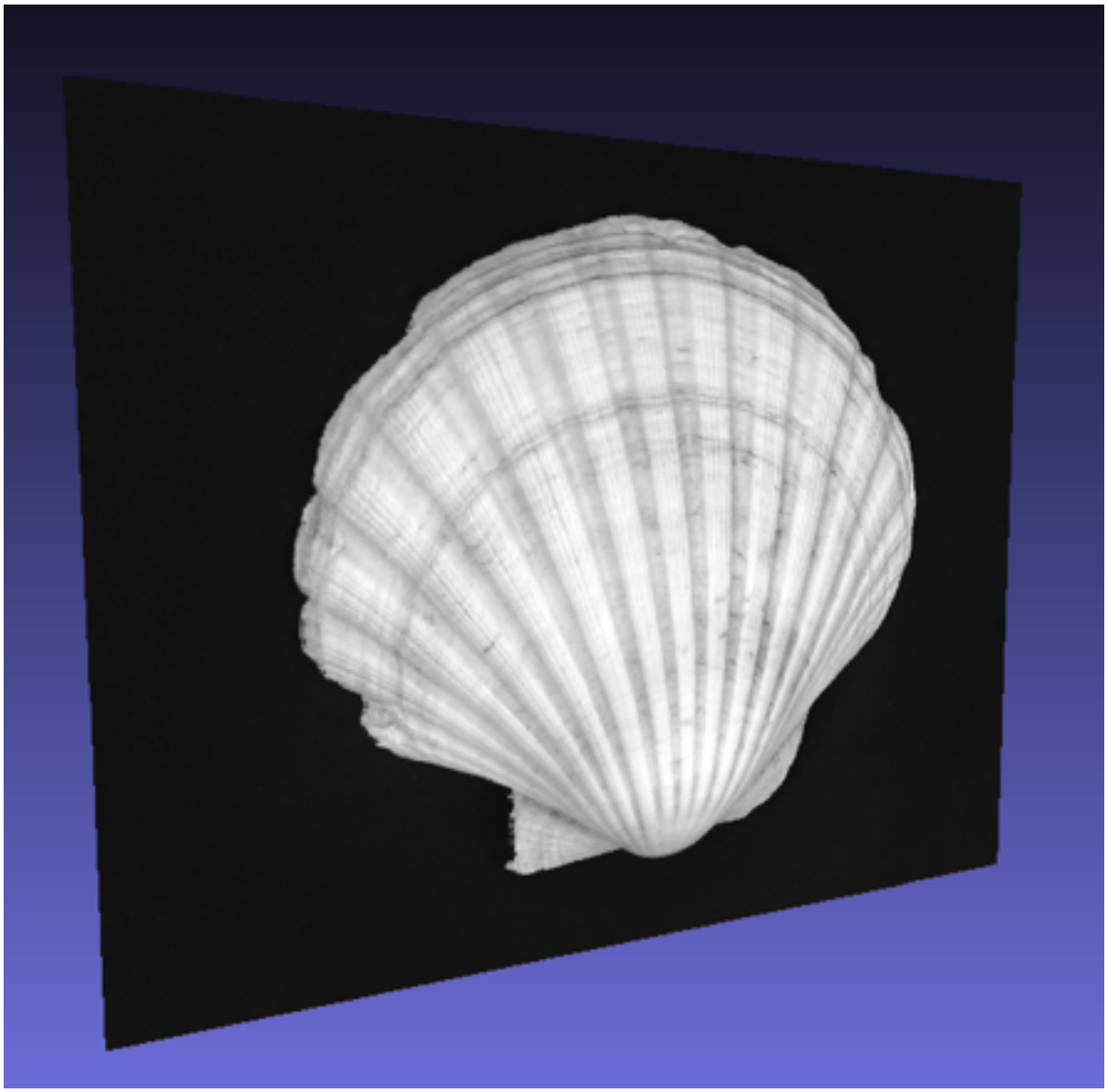

Figura 4.7: La superficie esportata, come viene visualizzata in un visualizzatore esterno

Capitolo 5

Conclusioni e sviluppi futuri

La realizzazione della GUI offre sicuramente uno spunto per favorire la diffusione di **ps3d**, rendendone molto meno complicato l'interfacciamento con l'utente finale. Un ulteriore passaggio potrebbe essere quello di compilare il pacchetto in un applicativo standalone per Windows, Linux e Mac OS in modo da non aver bisogno di un'installazione di MATLAB nella macchina host. L'applicativo di base è modulare e si è rivelato essere molto portato per accogliere una GUI, e nel corso dello sviluppo non ho riscontrato nessuna difficoltà non arginabile con facilità. La Photometric Stereo è una tecnica computazionalmente efficiente e dai costi di implementazione contenuti, ma che presenta diversi limiti: infatti la ricostruzione non produce un modello completo, ma un semimodello (ottenuto tagliando virtualmente l'oggetto da ricostruire con un piano longitudinale). Per risolvere questa problematica si possono ripetere *n* ricostruzioni utilizzando la PS per poi ricomporre l'oggetto finale, realizzando la cosidetta tecnica *Multi View*. Un'ulteriore miglioria potrebbe essere rappresentata dall'interfacciamento con *devices* mobili o fotocamere: un caso di studi realistico in cui questo potrebbe rappresentare un effettivo vantaggio è quello delle ricerche archeologiche, in cui l'utilizzo della Photometric Stereo diventa a volte un utile supporto; questo permetterebbe di acquisire in real time le immagini da elaborare per la ricostruzione delle superfici. Il software realizzato e descritto in questo elaborato è completamente Open Source.

Elenco delle figure

Bibliografia

[1] Woodham, R. J. (1980). Photometric method for determining surface orientation from multiple images. Optical engineering, 19(1), 191139.

[2] Rodriguez, G. (2008). Algoritmi numerici. Pitagora Editrice Bologna.

[3] Seatzu, S., Contu, P. (2012). Equazioni alle Derivate Parziali (Una introduzione ai metodi di risoluzione analitica e numerica) (pp. 1-245). Pitagora.

[4] R. Klette, K. Schlüns, and A. Koschan, Computer Vision: Threedimensional Data from Images. Singapore: Springer, 1998.

[5] A. Concas, R. Dessı, C. Fenu, G. Rodriguez, and M. Vanzi. Identifying the lights position in photometric stereo under unknown lighting. In Proceedings of the International Conference on Computational Science and its Applications (ICCSA), Cagliari, Italy, September 2021

[6] P. N. Belhumeur, D. J. Kriegman, and A. L. Yuille, "The bas-relief ambiguity," Int. J. Comput. Vis., vol. 35, no. 1, pp. 33–44, 1999

[7] Golub, Gene H.; Van Loan, Charles F. (1996). Matrix Computations (3rd ed.). Baltimore: Johns Hopkins. ISBN 978-0-8018-5414-9.

[8] Dessì, R., Mannu, C., Rodriguez, G., Tanda, G., Vanzi, M. (2015). Recent improvements in photometric stereo for rock art 3D imaging. Digital Applications in Archaeology and Cultural Heritage, 2(2-3), 132-139.